# Network Related Settings

The Servoy Application Server exposes several services to the network it is connected to. Among other they are the following Services:

- Launch & run Smart Clients
- Launch & run Web Clients
- Host the Servoy Admin page
- Expose services of plugins, like the PDF Forms and RESTful Web Services plugins

This chapter discusses the various configuration options in the area of network connectivity, including the ports over which the services are exposed and enabling HTTPS & SSL. The network configuration options for Smart Client connectivity comprises the majority of this chapter, as it is the most extensive.

## In This Chapter


- High Level Overview
- Setting the HTTP Port
- Enabling HTTPS
- Enforcing HTTPS
  - Enforcing HTTPS for All Traffic
  - Enforcing HTTPS on Selected Traffic
- Smart Client Specific Configuration
  - Connection Modes
    - Available Connection Modes
    - Resource Usage per Connection Mode
    - Setting the Connection Mode
    - RMI Port
    - Setting the RMI Port
  - RMI Server Hostname
  - RMI Connection Timeout
  - Ping Delay
  - Profiles
  - SSL Encryption
    - Setting Up SSL Encryption
  - Compression
    - Enabling/Disabling Compression
  - Running Smart Clients Behind a Proxy
  - Advanced HTTP Tunnel Configuration
    - Default HTTP Tunnel Configuration
    - Custom HTTP Tunnel Configuration
    - Suggested Profiles
  - Kerebos Support


## High Level Overview

The Servoy Application Server exposes the majority of it's services over the so-called **HTTP Port** (default port 8080). Through this port the Servoy Admin page, the Web Clients and plugin services are exposed to the outside world.

How Smart Clients communicate with the Servoy Application server depends largely on the chosen configuration. By default the communication goes through the so-called **RMI port** (default port 1099), but the Servoy Application Server can be configured to tunnel all the communication over the HTTP port as well, through the so-called **Tunnel**.

While Smart Clients communicate with the Application Server through the RMI port (or optionally the HTTP Tunnel) while it is running, the Smart Client is launched over the HTTP port.

All network communication with the Servoy Application Server can be optionally secured, by enabling HTTPS for all traffic over the HTTPS port and by enabled SSL encryption for the communication between the Smart Client and the Application Server.

## Setting the HTTP Port

The HTTP port, used to expose many of the services of the Servoy Application Server can be configured by editing the `server.xml` file located in `../application_server/server/conf`. This file contains the following entry by default:

```
<Connector
   port="8080"
   protocol="HTTP/1.1"
   maxThreads="500"
   connectionTimeout="60000"
   redirectPort="8443"
   useBodyEncodingForURI="true"
/>
```

By altering the value of the 'port' attribute, for example to 9090 or 80, the port on which the services of the Servoy Application Server are exposed can be changed. In order for the changes to go into effect a restart of the Application Server is required.

⚠ Note that on some operating systems, like Linux of FreeBSD, binding a process to a port number lower than 1024 (for example the default HTTP port 80) required the process to run as root or under administrator privileges.

# Enabling HTTPS

The Tomcat server that underlies the Servoy Application Server can be configured to support HTTPS. The Tomcat server that comes bundled with Servoy by default is not setup to support HTTPS, for two reasons:

1. Properly enabling HTTPS requires a keystore with signed SSL certificates that must be created externally from the installation process of the Servoy Application Server
2. In many network scenario's the Servoy Application Server is running behind another Web server that takes care of the HTTPS support.

If direct HTPPS support on the Servoy Application Server is required, it can be enabled in Tomcat by adding an additional connector, configured for secure access to the server.xml file located in `../application_server/server/conf`.

```
<Connector port="8443"
   maxThreads="500"
   connectionTimeout="60000"
   scheme="https"
   secure="true"
   SSLEnabled="true"
   keystoreFile="conf/keystore"
   keystorePass="changeit"
/>
```

In order to create a secure HTTPS connector a keystore with a signed SSL Certificate is required. The created keystore needs to be added to the Tomcat server installation that is part of the Servoy Application Server, located in `../application_server/server`. It's a best practice to place the keystore in the `{servoyInstall}/application_server/server/conf/` directory. Note that the same keystore can be used to encrypt the traffic between Smart Clients and the Servoy Application Server. See SSL Encryption for more details.

ℹ For more information on how to create a keystore, see Creating a keystore with a signed certificate.

⚠ While it's possible to enable SSL without a keystore, this is insecure and browsers will generate security warnings when accessing webpages through HTTPS in such scenario.

In the definition of the connector above the value of the following attributes need to be adjusted to be correct for the supplied keystore:

- **keystoreFile**: for example 'conf/mykeystore.ks'
  The value of this attribute needs to refer to the location and name of the keystore (either absolute or relative to the Tomcat server home directory (`../application_server/server`))
- **keystorePass**: for example 'mypassword'
  The passPhrase of the keystore. The passPhrase is specified when creating the keystore

Additionally, the value of the port attribute needs to be brought in sync with the value of the redirectPort attribute of the standard HTTP connector (or vise versa), as the redirectPort attribute on the HTTP connector is used to redirect HTTP traffic to HTTPS when required, see Enforcing HTTPS for all traffic. The value for the port can be any value. By default the redirectPort number on the HTTP Connector is set to 8443, but any value, including the default HTTPS port 443 is possible.

Note that on some operating systems, like Linux of FreeBSD, binding a process to a port number lower than 1024 (for example the default HTTPS port 443) requires the process to run as root or under administrator privileges.

Lastly, to prevent issues with Internet Explorer 8, the Tomcat server needs to be configured to not add a 'pragma:no-cache' header to all requests. To setup Tomcat to not do this, create a file called context.xml in the `../application_server/server/conf` directory with the following content:

```
<!-- The contents of this file will be loaded for each web application -->
<Context>

<!-- Default set of monitored resources -->
<WatchedResource>WEB-INF/web.xml</WatchedResource>

<!-- Uncomment this to disable session persistence across Tomcat restarts -->
<!--
<Manager pathname="" />
-->

<Valve className="org.apache.catalina.authenticator.BasicAuthenticator" disableProxyCaching="false"/>
<Valve className="org.apache.catalina.authenticator.NonLoginAuthenticator" disableProxyCaching="false"/>
<Valve className="org.apache.catalina.authenticator.FormAuthenticator" disableProxyCaching="false"/>
<Valve className="org.apache.catalina.authenticator.DigestAuthenticator" disableProxyCaching="false"/>
<Valve className="org.apache.catalina.authenticator.SSLAuthenticator" disableProxyCaching="false"/>
</Context>
```

# Enforcing HTTPS

If HTTPS is enabled, it's possible within the Tomcat server underlying the Servoy Application Server to redirect specific or all incoming HTTP traffic to HTTPS by editing the `web.xml` file located in `../application_server/server/webapps/ROOT/web_inf`.

## Enforcing HTTPS for All Traffic

To redirect all HTTP traffic to HTTP, add the following security-contraint at the the bottom of the file, just before '`</web-app>`':

```
<security-constraint>
    <web-resource-collection>
      <web-resource-name>Automatic SLL Forwarding</web-resource-name>
      <url-pattern>/</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
```

When forcing all HTTP requests to HTTPS, the `servoy.jnlpCodebaseOverride` setting needs to be the HTTPS URL (including the HTTPS Connector port number).

## Enforcing HTTPS on Selected Traffic

Instead of redirecting all traffic to HTTPS it's also possible to redirect only specific traffic to HTTPS or exclude specific traffic. Which traffic is redirected or not is controlled by the <url-pattern> and/or <url-pattern-exclude> nodes of the <web-resource-collection> node of the security-constraint.

All url mappings used by Servoy can be found in the file web.xml located in `../application_server/server/webapps/ROOT/WEB-INF`.

To force HTTPS only on the Servoy Admin page for example the following security-constraint ought to be used:

```
<security-constraint>
    <web-resource-collection>
      <web-resource-name>Automatic SLL Forwarding</web-resource-name>
      <url-pattern>/servoy-admin/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
```

To force HTTPS on all Web Client traffic and the Servoy Admin page, but excluding it for Template access, the following security-constraints are to be used:

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Unsecure access</web-resource-name>
    <url-pattern>/servoy-webclient/templates/*</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint>

<security-constraint>
  <web-resource-collection>
    <web-resource-name>Automatic SLL Forwarding</web-resource-name>
    <url-pattern>/servoy-admin/*</url-pattern>
    <url-pattern>/servoy-webclient/*</url-pattern>
    <url-pattern>/servoy-webclient</url-pattern>
  </web-resource-collection>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

# Smart Client Specific Configuration

The network configuration options for Smart Clients are quite extensive and which configuration to choose is largely dependent on the (different) network setups between the Servoy Application Server and the machines on which Smart Clients are launched. Determining the most optimal network configuration for Smart Client comes down to answering the following questions:

- Can all client machines access the RMI port on the Servoy Application Server?
- Can the Servoy Application Server directly access the client machines on any port?
- Can all client machines access the Servoy Application Server on the same IP address?
- Are some of the client machines configured to access webpages through a proxy?

The answers to these questions could eliminate one or more of the four possible Connection Modes, the the matrix below:

| | Direct Connection | Two-Way Socket | HTTP Tunnel | Socket Tunnel |
|---|---|---|---|---|
| Client machines require direct access to the Application Server's HTTP port | ✓ | ✓ | ✓ | ✓ |
| Client machines require direct access to the Application Server's RMI port | ✓ | ✓ | ✗ | ✓ |
| Application Server requires direct access to all ports on each client machine | ✓ | ✗ | ✗ | ✗ |
| Supports client machines with Web Start proxy configuration | ✓ | ✗ [1] | ✓ | ✓ |
| Supports multiple IP addresses for the Application Server | ✗ | ✓ | ✓ | ✓ |

[1] *Two-Way socket mode* cannot initialize properly if Java WebStart on the Client machine is configured to use a Proxy. When it fails to initialize, the Smart Client will fall back to Direct Connection mode. See 'Two-Way socket mode' under Connection Modes below for more details.

## Connection Modes

The Servoy Application Server has several modes in which Smart Clients can communicate with the Application Server. Which mode is the best depends on the network setup between the Servoy Application Server and the client machines on which the Smart Client will be launched. As the Servoy Smart Client runs over both a LAN and WAN's, including over the internet, it can be that there are different network setups for different client machines.

### Available Connection Modes

*Two-Way socket*

> *Two-Way socket mode provides a robust communication mechanism between Smart Clients and the Servoy Application Server, where only the Smart Client initiates connections to the Application Server over the RMI port. This means that only the Smart Clients need to be able to access the Application Server and that the Application Server does not need to be able to connect to the client machine, like is required when using Direct Connection mode.*
> *However, in case Java WebStart on the client machine is configured to connect through a proxy, Servoy will not be able to instantiate Two-Way socket and thus falls back to the Direct Connection mode, with the restrictions that come with Direct Connection mode. It is possible to configure Java WebStart on each client machine to not use a proxy, but this needs to be done on each individual machine and might conflict with other Java WebStart applications that do require the proxy settings. If this happens then on the client the java preferences > network settings need to be changed, so that the proxy settings are configured as 'direct connection' instead of 'use browser'. Better is to use one of the tunnels below.*

> *Within individual Smart Clients it's possible to turn off Two-Way socket through Preferences. When the user does so, the Smart Client will start to use Direct Connection. The Smart Client will error if the network criteria for Direct Connection mode to work are not met.*
>
> *Note that disabling Two-way socket mode on the Application Server is pushed to all Smart Client, but when re-enabling Two-way socket on the Application server, this setting is NOT pushed to the Smart Clients. The user needs to manually enable TwoWay socket under Preferences in the Smart Client again.*

*Tunnel*

> *Servoy comes with a so-called tunnel that Smart Clients can use to connect to Application Server. The tunnel is a more robust communication mode, at the cost of utilization of more server-side resources. For each Smart Client more memory, threads and sockets (connections) are used on the Application Server. The tunnel supports two modes, namely HTTP and Socket which can be used exclusively or mixed through the use of Profiles. Of the two tunnel modes, the HTTP mode consumes most serverside resources.*
>
> *HTTP Tunnel*
> *When using the tunnel in HTTP mode, all communication between the Smart Client and the Application Server is done using the HTTP Protocol, over the HTTP port on which the Application Server runs. The benefit of this mode is that it doesn't require that the client machines can access the Application Server on the RMI port. This can be a huge benefit in cases where it's not possible to open up access to the RMI port. When Smart Clients connect to the Application Server using the HTTP Tunnel, they will utilize threads and connections from the Tomcat application server that underlies the Servoy Application Server.*
>
> *Socket Tunnel*
> *The tunnel in Socket mode is similar to the tunnel in HTTP mode, except that the communication with the Application Server goes over the RMI port of the Application Server. This means that the RMI port of the Application Server should be accessible from all client machines that will run Smart Clients. Unlike the tunnel in HTTP mode, the tunnel in Socket mode uses server-side resources (threads & sockets (connections)) directly in Servoy, not through Tomcat.*

### Resource Usage per Connection Mode

The overview below indicated the difference in used resources per Connection mode. This information is provided in case tuning is required for large scale deployments. For example, by default the Tomcat HTTP/HTTPS connectors are configured to allow a maximum of 500 concurrent threads. When deploying large numbers of Smart Clients over the HTTP tunnel (possible in combination with a large number of Web Clients), it might be required to change the maximum allowed threads in the Tomcat connectors.

| Connection Mode | Used Threads | Used sockets |
|---|---|---|
| Two-Way socket | 1 thread per concurrent request | 1 per request + 2 per client |
| HTTP Tunnel | 2 Tomcat and 2 non-tomcat threads per client | 2 per client |
| Socket Tunnel | 2 threads per client | 1 per client |

### Setting the Connection Mode

All connection modes can be configured through the Servoy Admin page, under **Network Settings**.

This can be done by the property: `SocketFactory.tunnelConnectionMode`

4 options can there be set: http, socket,http&socket and 2waysocket.

The preferred way is http&socket, this means that both http tunnel and socket tunnel are enabled, by default a client will then connect in socket mode but you can have a profile so that the client will start in http mode.

*Two-Way socket*

> *SocketFactory.useSSL: set to true [1]*
> *SocketFactory.compress: set to true*
>
> *If the server on which the Servoy Application Server runs has multiple network interfaces through which Smart Clients ought to connect:*
> *java.rmi.server.hostname: set to the IP address of the server's loopback interface, usually 127.0.0.1*

*Tunnel*

> *SocketFactory.useSSL: set to true [1]*
> *SocketFactory.compress: set to true*
>
> *SocketFactory.tunnelUseSSLForHttp: set to false if tomcat is configured in https else true, see SSL Encryption*

[1]: SSL can also be turned off, but for security reasons it is advised to have SSL Encryption turned on when possible.See SSL Encryption for additional settin gs.

### RMI Port

With all connection modes, except the HTTP Tunnel, the Smart Clients communicate with the Servoy Application Server over the so-called RMI port. This port needs to be accessible from all the client machines at all times, thus cannot be blocked by firewalls.

Through configuration the RMI Start Port value can be set. When the Servoy Application Server is launched it tries to bind to the specified RMI Start port. If it fails to bind to this port, it will automatically try to bind to the next port. This process will continue until the binding succeeds. The actually used RMI port should never be blocked by any firewall in between the Servoy Application Server and the client machines.

The RMI port number that is actually used is shown under **Server Information** in the **Servoy Server Status** on the Servoy Admin page under **Servoy Server Home**

#### Setting the RMI Port

The RMI Start port can be set through the Servoy Admin page under **Network Settings** > **servoy.rmiStartPort**. Default value is 1099

## RMI Server Hostname

The meaning of the RMI Server Hostname setting (`java.rmi.server.hostname`) depends on the Connection Mode used:

| Connection Mode | Relevance |
|---|---|
| Two-Way Socket | When the machine of which the Servoy Application Server runs has multiple network interfaces on which the Smart Clients could connect, this setting needs to be set to the IP address of the server's loopback interface (usually 127.0.0.1) |
| HTTP/Socket Tunnel | no relevance |

⚠ If a hostname is specified, note that the hostname is resolved to an IP address on the server!

## RMI Connection Timeout

The RMI Connection timeout setting (rmi.connection.timeout) controls how long a the RMI layer will hang on to open sockets for reuse. A high value can lead to many open sockets.

If there are a high number of open sockets, it can happen that a network component like a Firewall or NAT Router automatically starts closing them. When this happens, when Servoy tries to use the socket connection, it will error. Servoy will automatically remove the erroneous socket and try the next one. This will cause a slight performance decrease.

By lowering the RMI Connection timeout value, the total number of open sockets will be reduced, preventing the network component from closing them automatically.

| Value | Comment |
|---|---|
| 120 | Default value |
| 10 ~ 15 | Alternative value to bring down the number of open sockets, to prevent the network component form automatically closing them |
| 0 | When set to 0, the system default will be used |

## Ping Delay

The Ping Delay setting (ApplicationServer.pingDelay) provides a mechanism to prevent network components from closing open connections that are inactive for a while.

When set too high, and a network component between the Application Server and the Smart Client closes the connection, the User will be presented with a Connection Lost dialog, from which the user can choose to reconnect or close the Smart Client.

When experiencing intermittent Connection Lost dialogs in the Smart Client, lower the ping delay.

When using the HTTP Tunnel exclusively for connecting Smart Clients, the alternative HTTP tunnel KEEP_ALIVE_INTERVAL can be used instead, as this option has a lower overhead than the `ApplicationServer.pingDelay` setting. The `ApplicationServer.pingDelay` can be set to a very high number in this case.

## Profiles

A Profile is a named set of settings that can be used by a Smart Client.

Profiles are a way to allow the definition of multiple sets of configurations that are used by groups of Smart Clients. For more information on Profiles, see Profiles.

## SSL Encryption

All the communication between the Servoy Application Server and the Smart Clients can be encrypted using SSL Keys and certificates. Servoy Smart Clients can only work with either a 3rd party signed certificates or without a certificate. Self signed certificates are not supported by default, but can be made to work by adding the setting `SocketFactory.useTestSSL` to either a Profile that is used or to the system.properties (**Admin page** > **Admin settings** > **system.properties**)

When SSL is enabled it will also encrypt all properties in the *servoy.properties* file that have a property key that contains the text 'password'. For the encryption the specified SSL certificate will be used (or a default certificate if no keystore with certificates is specified). This means that if the certificate /keystore is changed or SSL is disabled, the passwords cannot be decrypted anymore and have to be manually set again in plain text in the servoy.properties file. If the encrypted properties are not reset manually, the Servoy Application Server will fail to launch.

### Setting Up SSL Encryption

Enabling SSL Encryption for communication between Smart Clients and the Servoy Application Server requires a keystore with a signed SSL certificate and the adjustment of a three settings on the Servoy Admin page. For more information on how to create a keystore, see Creating a keystore with a signed certificate .

The created keystore needs to be added to the Servoy Application Server installation. Best practice is to place the keystore in the `../application_server/server/conf` directory. In this location the keystore is then also available to the Tomcat server underlying the Servoy Application Server and thus the same certificate can also be used for serving HTTPS content (see Enabling HTTPS).

After making the keystore available to the Servoy Application Server, the Servoy Application Server needs to be told where the keystore can be found and configured to use it. The relevant settings are exposed under the Network Settings on the Servoy Admin page:

- SocketFactory.useSSL: set to true
- SocketFactory.tunnelUseSSLForHttp: if the tunnel is in http or http&socket mode you can specify if http traffic should also be encrypted by the factory. If clients already only connect through HTTPS this setting can be set to false. Because the content is already encrypted by that connection done by tomcat.
- SocketFactory.SSLKeystorePath: set to the location and name of the keystore. The path must be relative to the Servoy Application Server installation directory (`../application_server/`), for example `server/conf/mykeystore.ks`
- SocketFactory.SSLKeystorePassphrase: Set to the passPhrase used when creating the keystore
- SocketFactory.SSLSupportedSuites: If you need to be complaint with 'PCI certification' or something like that. Or you really don't want to enable weak cyphers between server and client. You can here specify which once you do want to support. See here what ciphers can be chosen from as an example and how tomcat can be configured also (the https part) http://www.sslshopper.com/article-how-to-disable-weak-ciphers-and-ssl-2-in-tomcat.html. This only works for the the sockets in 'http&socket' or just 'socket' configurations.

Note that SSL can also be enabled without specifying the keystore, by just setting `SocketFactory.useSSL` to *true*. This is considered insecure and should only be used for testing/demo purposes.

## Compression

Compression reduces the amount of data send back and forth between the Servoy Application Server and the Smart Clients, thus improving performance.

### Enabling/Disabling Compression

The Compression setting can be administered through the Servoy Admin page, under **Network Settings** > **SocketFactory.compress**. Compression is turned on by default and should always be on. It should only be disabled when experiencing connection issues due to compression.

## Running Smart Clients Behind a Proxy

In many scenario's there will be proxies in between the Smart Client and the Application Server. Depending on the connection mode through which the Smart Client connects and the configuration of Java WebStart on the client machine, this will either not be an issue at all, or require some configuration or will not work.

Proxy configuration can be done in Java WebStart on the client machine.

| Connection mode | Proxy Supported? | Config | Comments |
|---|---|---|---|
| Two-Way Socket | No | N/A | Two-way socket communication cannot be instantiated by the Smart Client inside Java WebStart if Java WebStart is configured to run through a proxy.<br>The Smart Client will fall back to Direct Connection mode, which might work, as it requires the Servoy Application Server to be able to reach each individual client machine an any random port |
| HTTP Tunnel | Yes | Not required by default | It's possible to explicitly specify the proxy url, username and password, through Advanced HTTP Tunnel configuration |
| Socket Tunnel | Yes | | No configuration required |

⚠ Note: up to and including Servoy 5.2.8 it is required to manually set the following system property on the Application Server to get the best possible: SocketFactory.useProxySelector=true. As of Servoy 5.2.9 and Servoy 6.0 this setting with be done by default

## Advanced HTTP Tunnel Configuration

In the HTTP Tunnel connection mode, all communication between the Smart Client and the Servoy Application Server is done through the HTTP protocol and as such is susceptible to the influences of network components and/or processes that handle the HTTP traffic. Such influences can be anti-virus software, firewalls, proxies, NAT routers etc. While all these influences in general serve a goal, they can pose a challenge when trying to establish a reliable connection between the Smart Client and the Application Server.

The negative influences that these network components and/or processes can cause are, at a high level, summarized below.

*HTTP Chunked mode & Buffering*

> *The HTTP protocol allows for the sending of HTTP requests in small chunks, instead of the entire request at once. As there is significant overhead on the instantiation of HTTP requests, utilizing the chunked modus to send a package of data as a chunk of a larger HTTP request instead of instantiating an HTTP request for each package of data provides a way to limit the overhead of HTTP request instantiation.*
>
> *However, some components in a network connection that deal with HTTP traffic perform buffering. This is the the process of holding onto individual chunks of the HTTP requests until the entire request is received, before sending the full request to the next component in the network connection. This means that the individual chunks only reach their destination when all the chunks that belong to an individual HTTP request are received.*

*Connection timeouts*

> *Components in the network connection over which an HTTP request is executed might inflict a limit on how long the connection may be open, before it is automatically closed.*

*Content-Length enforcement*

> *The HTTP protocol contains a mechanism to indicate the length of an HTTP request. While is not common practice to check the supplied length with the actual length and reject the request if those do not match, it can be that there are components in the network connection between the Smart Client and Application Server that do perform such a check and reject the HTTP requests if the numbers do not match.*
>
> *Additionally some components in the network might have a maximum content length that they enforce.*

### Default HTTP Tunnel Configuration

In everyday life, buffering is the most likely candidate to cause issues, thus by default the HTTP tunnel is configured to NOT use Chunked mode, at the cost of performance. The HTTP Tunnel is also configured by default with values for the relevant settings to prevent Connection timeout issues in most scenario's. In the area of content length enforcement, the maximum length for HTTP requests is set to a proper value. The enforcing of the actual content length being the same as the specified content length is NOT done, as the likelihood of network connections enforcing this not that great, while enabling this would result in significant performance degradation.

### Custom HTTP Tunnel Configuration

The configuration of the HTTP tunnel can be altered by adding the relevant settings as system properties through the Servoy Admin page, under **Servoy Server Home** > **Admin settings** > **system.properties**. It's also possible to expose different configurations through Profiles.

The following settings are available:

| Property | Default Value | Description |
|---|---|---|
| system.property.com.sebster.tunnel.http.client.closeRequestOnFlush | true | Whether or not to close the HTTP Request after each package of data. Default is true. While this incurs a significant overhead due to having to instantiate an HTTP request for each package of data, this does prevent buffering issues.<br>Besides incurring significant overhead, it also results in a large number of HTTP Requests to the Application Server, which at some point might start blocking the requests. |
| system.property.com.sebster.tunnel.http.client.chunked | true | Whether or not to send data packages as chunks of a larger HTTP Request. This setting has no effect if closeRequestOnFlush is set to true. Chunked mode is the preferred mode of operation, but susceptible to buffering issues. |
| system.property.com.sebster.tunnel.http.client.keepAliveInterval | 15 | The interval (in seconds) with which a little data is send into the HTTP request to prevent it from being terminated by the network due to inactivity. Lower when experiencing connection timeout issues. |
| system.property.com.sebster.tunnel.http.client.maxRequestAge | 300 | The maximum time (in seconds) that a HTTP request is kept alive. Lower when experiencing timeout issues. |
| system.property.com.sebster.tunnel.http.client.contentLength | 2097152 b | The maximum length of a single HTTP Request. Lower only when the network enforces a lower limit |

| | | |
|---|---|---|
| system.property.com.sebster.tunnel.http.client.strictContentLength | false | If set to true, the actual length of each HTTP request will be made to match the contentLength setting, by appending dummy data. Only set to true if the Content-Length is enforced by the network, as it incurs a significant performance penalty |
| system.property.com.sebster.tunnel.http.client.proxyUri | | In case the Smart Client needs to communicate through a specific proxy and the proxy configuration cannot be done in Java WebStart, the proxy url can be set directly on the HTTP Tunnel. |
| system.property.com.sebster.tunnel.http.client.proxyUsername, com.sebster.tunnel.http.client.proxyPassword | | Proxy credentials. If required by the proxy and not set here, the Smart Client will provide the user with a dialog to enter the credentials |
| system.property.com.sebster.tunnel.http.client.userAgent | | Empty by default. Can be set to any value if it is required to send a specific user-agent string with each HTTP request |

### Suggested Profiles

The HTTP tunnel is already configured for most reliable connectivity, so this does not require a Profile. A relevant Profile would be the performance Profile, which basically means enabling Chunked mode, by disabling 'closeRequestOnFlush'

**Best performance Profile**

```
system.property.com.sebster.tunnel.http.client.CLOSE_REQUEST_ON_FLUSH=false
```

⚠️ Note that the example above assume that the HTTP tunnel is already enabled. See Profiles for more info.

## Kerebos Support

Servoy supports Kerebos authentication for Smart Clients, as Java WebStart supports Kerebos authentication. Setting up Kerebos authentication within Java WebStart on the client machine will not be described in the Servoy documentation, as it's highly specific and requires detailed knowledge of Kerebos configuration.

If Kerebos authentication support is required, the following setting sets up required authentication context within Servoy: system.property.servoy.usejaas=true. This setting can be applied through the Servoy Admin page, under **Servoy Server Home** > **Admin settings** > **system.properties** or can be applied through a specific Profile.

For the configuration of Java WebStart on the client machines, please contact a Kerebos specialist.

⚠️ Kerebos authentication is currently not supported in combination with the HTTP and Socket Tunnel